# MIEIC – Programação – 1.º ano

# PROJETO Nº2 – 2011/2012

# Definição de classes a utilizar

# *Duarte Duarte & Miguel Marques*

## AllegroObject

Classe abstract que define um objecto que pode ser desenhado no ecrã

**public**

AllegroObject() { } // Construtor

**virtual** **void** Draw() = 0; // Desenho no ecrã

**virtual** **bool** Update(ALLEGRO\_EVENT\* ev) = 0; // Atualização de estado

## RectButton: AllegroObject

Define um botão clicável com várias opções de estilo. Quando clicado, o botão poderá chamar uma função (callback).

**public**

RectButton(Vector2D size, Vector2D position, ALLEGRO\_COLOR color, ALLEGRO\_COLOR colorMouseHover, ALLEGRO\_COLOR textColor, std::string text, uint fontSize, **bool** func(RectButton\*), **bool** shadowedText); // Botão com todas as opções

RectButton(Vector2D size, Vector2D position, ALLEGRO\_COLOR color, ALLEGRO\_COLOR colorMouseHover, **bool** func(RectButton\*) = NULL); // Botão sem texto com “mouse hover”

RectButton(Vector2D size, Vector2D position, ALLEGRO\_COLOR color, ALLEGRO\_COLOR textColor, std::string text, uint fontSize, **bool** func(RectButton\*), **bool** shadowedText = **false**); // Botão com texto sem “mouse hover”

RectButton(Vector2D size, Vector2D position, ALLEGRO\_COLOR color, **bool** func(RectButton\*) = NULL); // Botão sem texto sem “mouse hover”

RectButton(Vector2D position, ALLEGRO\_COLOR textColor, std::string text, uint fontSize, **bool** func(RectButton\*), **bool** shadowedText = **false**); // Botão transparente com texto

RectButton(Vector2D size, Vector2D position, **bool** func(RectButton\*) = NULL); // Botão transparente sem texto

**bool** Update(ALLEGRO\_EVENT\* ev); // Atualizaçao do estado

**void** Draw(); // Desenho no ecrã

**void** Draw(**bool** forcedMouseHovered) // Desenho no ecrã com “mouse hovered” forçado

**bool** IsMouseHovered(); // O ponteiro do rato está sob o botão?

std::string GetText(); // Texto central

**private**

Vector2D \_size; // Tamanho do botão

Vector2D \_position; // Posição do botão

std::**string** \_text; // Texto central

bool \_forcedMouseHovered; // "Mouse hovered" forçado

bool \_clicked; // Foi clicado?

bool \_shadowedText; // Text com sombra?

uint \_fontSize; // Tamanho da letra

ALLEGRO\_COLOR \_color; // Cor do botão

ALLEGRO\_COLOR \_colorMouseHover; // Cor do botão quando "mouse hovered"

ALLEGRO\_COLOR \_textColor; // Cor do texto

bool (\*\_func)(RectButton\*); // Função chamada quando botão é clicado

## Fonts

Gere os tipos de letra utilizados pela aplicação

**public**

**static** **void** UnloadFonts(); // Destroí os recursos usados

**static** ALLEGRO\_FONT\* GetFont(uint size); // Retorna uma fonte do tamanho especificado

**private**

static std::unordered\_map<uint, ALLEGRO\_**FONT**\*> \_fonts; // Mapa que associa cada tamanho de cada tipo de letra à sua ALLEGRO\_FONT respectiva

## Localization

O programa suporta várias línguas sendo cada uma carregada a partir de ficheiros binários codificados em UTF-8 (suporte de caracteres especiais) As frases nos ficheiros de línguas devem estar seperadas por '/0' (null char)

**public**

static Localization\* Instance(); // Obtém o endereço único **do** objecto

std::vector<Language> GetAvailableLanguages() **const**; // Retorna um vector das línguas disponíveis

std::**string** GetString(**String**s index) **const**; // Retorna uma frase/palavra na língua especificada em SetLang()

void SetLang(Language lang); // Especifica a língua que irá ser usada

**private**

Localization(); // Construtor privado

**static** Localization\* \_instance; // Endereço único do objecto

std::vector<Language> \_languages; // Línguas disponíveis

std::vector<std::string> \_strings; // Conjunto de frases para a língua selecionada

Language \_currLang; // Língua selecionada

**bool** ReadLangFile(**char**\* lang); // Lê ficheiro binário de cada língua individual (por exemplo, pt.lang)

**bool** FindLangs(); // Procura ficheiros de línguas no directório em que se encontra o programa

**static** Language GetLanguageByShortLang(std::string lang); // Converte, por exemplo, de "pt" para "Portuguese"

## InvalidCardException: std::exception

Exceção de carta inválida

**public**

InvalidCardException(std::string message = "Invalid Card!") // Construtor

~InvalidCardException() **throw**() {} // Destrutor

**const** **char**\* what() **const** **throw**() { **return** \_message.c\_str(); } // Mensagem de erro

**private**

std::string \_message; // Mensagem de erro

## InvalidScoreException: std::exception

Exceção de pontuação inválido

**public**

InvalidScoreException(std::string message = "Invalid Score!") // Construtor

~InvalidScoreException() **throw**() {} // Destrutor

**const** **char**\* what() **const** **throw**() { **return** \_message.c\_str(); } // Mensagem de erro

**private**

std::string \_message; // Mensagem de erro

## InvalidPlayerException: std::exception

Exceção de jogador inválido

**public**

InvalidPlayerException(std::string message = "Invalid Player!") // Construtor

~InvalidPlayerException() **throw**() {} // Destrutor

**const** **char**\* what() **const** **throw**() { **return** \_message.c\_str(); } // Mensagem de erro

**private**

std::string \_message; // Mensagem de erro

## InvalidBetException: std::exception

Exceção de aposta inválida

**public**

InvalidBetException(std::string message = "Invalid Bet!") // Construtor

~InvalidBetException() **throw**() {} // Destrutor

**const** **char**\* what() **const** **throw**() { **return** \_message.c\_str(); } // Mensagem de erro

**private**

std::string \_message; // Mensagem de erro

## FileNotFoundException: std::exception

Exceção de ficheiro não encontrado

**public**

FileNotFoundException(std::string filename, std::string message = "File not found!") : \_message(message), \_filename(filename) {} // Construtor

~FileNotFoundException() **throw**() {} // Destrutor

**const** **char**\* what() **const** **throw**() { **return** \_message.c\_str(); } // Mensagem de erro

**const** **char**\* filename() **const** **throw**() { **return** \_filename.c\_str(); } // Nome do ficheiro não encontrado

**private**

std::string \_message; // Mensagem de erro

std::string \_filename; // Nome **do** ficheiro não encontrado

## BlackJack

Faz "handling" dos estados do programa (menu principal/jogo/opções/fim do jogo) e contém o ciclo central do jogo (enquanto !terminado -> update, draw; repete)

**public**

**static** BlackJack\* Instance(); // Acesso estático ao objecto (apenas existirá um, "singleton")

**static** Vector2D GetMousePosition(); // Posição actual do rato

**void** \_Start(); // Início de todo o processo

**void** Quit(); // Encerramento do programa

**void** ChangeState(**int** newState); // Muda o estado

ALLEGRO\_DISPLAY\* GetDisplay(); // Acesso ao "display" (usado por Allegro)

**private**

BlackJack(); // Construtor privado

**static** BlackJack\* \_instance; // Pointer para o endereço único do objecto

**static** ALLEGRO\_MOUSE\_STATE\* \_mouseState; // Estado do rato (x,y, botões clicados, etc.)

**void** Initialize(); // Inicialização de algumas variáveis, chamado depois do construtor

**void** LoadContents(); // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**void** Update(ALLEGRO\_EVENT\* ev); // Chamado cada vez que há um evento novo (ver \_eventQueue); actualiza o estado

**void** Draw(); // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**void** UnloadContents(); // "Descarrega" sons, imagens e outros objectos

ALLEGRO\_DISPLAY\* \_display; // "display" (usado por Allegro)

ALLEGRO\_EVENT\_QUEUE\* \_eventQueue; // fila de espera de eventos (teclado, rato, temporizador e "display")

ALLEGRO\_TIMER\* \_timer; // temporazidor (configurado para "disparar"(trigger) a cada 1/60 secs

**int** \_state; // estado actual

**bool** \_done; // programa continua a ser actualizado enquanto \_done for falso

**bool** \_draw; // define se é necessário re-desenhar o ecrã

std::vector<State\*> \_states; // contentor dos estados

## Card // : AllegroObject

Define uma carta

**public**

Card(**int** suit, **int** rank); // Construtor

**int** GetSuit() **const**; // Naipe da carta (paus, ouros, ...)

**int** GetRank() **const**; // Tipo da carta (ás, dois, ...)

uint GetScore() **const**; // Valor da carta

**float** GetFrameWidth() **const**; // Largura da carta no ecrã

**float** GetFrameHeight() **const**; // Altura da carta no ecrã

**void** SetScore(**int** score); // Altera o valor da carta (para os ases)

**bool** IsValid() **const**; // A carta é válida?

**void** Draw(**float** dx, **float** dy, **float** angle = 0.0, **bool** mouseHovered = **false**); // Desenha a carta no ecrã na posição especificada; ângulo em radianos

**void** DrawBack(**float** dx, **float** dy, **float** angle = 0.0); // angle must be in radians // Desenha a parte de trás da carta no ecrã na posição especificada; ângulo em radianos

**bool** IsMouseHovered() **const**; // O ponteiro do rato está sob a carta?

**static** **void** DestroyBitmaps(); // Destroí as imagens usadas pela carta da memória

**private**

**int** \_suit; // Naipe da carta (paus, ouros, ...)

**int** \_rank; // Tipo da carta (á**s**, dois, ...)

uint \_score; // Valor da carta

**int** \_backColorRGB; // Cor usada pela parte de **tr**á**s** da carta ("tinted \_backImage")

bool \_isMouseHovered; // O ponteiro **do** rato está sob a carta?

static ALLEGRO\_BITMAP\* \_image; // Imagem frontal da carta; nota: é usada uma imagem que conté**m** todas as cartas (tileset) em que a posição da carta actual é calculada atravé**s** **do** naipe e tipo (4 linhas, 13 colunas)

static ALLEGRO\_BITMAP\* \_backImage; // Parte de **tr**á**s** da carta

static Vector2D \_frameSize; // Tamanho **do** desenho

## Deck // : AllegroObject

Define um baralho de cartas (ou vários baralhos)

**public**

Deck(uint numberOfDecks = DEFAULT\_NUMBER\_OF\_DECKS); // Construtor **do** baralho

Card\* WithdrawCard(); // Retira uma carta **do** topo **do** baralho

uint GetGameCardsNumber(); // Número de cartas existentes **no** baralho

void Draw(float dx, float dy, bool cardBack = false); // Desenha o baralho **no** ecrã

**private**

std::vector<Card> \_cards; // Vector de cartas

void InitializeDeck(uint numberOfDecks); // Insere no valor todos os tipos e naipes (ordenados)

void Shuffle(); // Baralha o baralho

## Hand // : AllegroObject

Define uma mão de cartas. Tanto usada pelos jogadores como pelo dealer

**public**

Hand(Vector2D position, **bool** dealerHand = **false**); // Construtor

Hand(); // Construtor

~Hand(); // Destrutor

uint GetScore() **const**; // Pontuação da mão

**bool** IsBusted() **const**; // Ultrapassou a pontuação limite?

**bool** IsBlackjack() **const**; // Fez blackjack?

Hand& AddCard(Card\* card); // Adiciona uma carta à mão

**void** RemoveCard(**const** Card\* card); // Tira uma carta à mão

**void** Clear(); // Tira todas as cartas à mão

**void** Draw(); // Desenha a mão de cartas

**void** ShowSecondCard(); // Mostra a segunda carta (usada pelo dealer quando recebe uma carta de valor 10 ou 11 como carta inicial)

**private**

std::vector<Card\*> \_cards; // Vector de pointers para as cartas da **m**ão

uint \_score; // Pontuação das cartas

bool \_dealerHand; // Esta é a **m**ão de cartas de um dealer?

bool \_drawSecondCardBack; // Deve desenhar a segunda carta voltada para baixo?

**int** \_cardJustAdded; // Nova carta acabou de ser adicionada?

Vector2D \_position; // Posição para o desenho

void UpdateScore(); // Re-calcula a pontuação da **m**ão

## Player // : AllegroObject

Define um jogador do jogo

**public**

Player(std::ifstream& file, S\_Game\* game); // Construtor

std::string GetName() **const**; // Nome do jogador

**double** GetBalance(); // Montante total do jogador

**double** GetBet() // Apostra realizada

**static** std::string GetPlayersFileName(); // Nome do ficheiro de jogadores (players.txt)

**void** WriteText(std::ofstream& out) **const**; // Guarda uma stream informações do jogador

// In-game player actions

**void** PlaceBet(**double** bet); // Faz uma aposta

**bool** Stand(); // Ficar

**bool** Hit(); // Pedir

**bool** Double(); // Dobrar

**void** Lose(); // Jogador perde

**void** ResetPlayer(); // Prepara o jogador para uma nova partida

// Hand related

**void** NewCard(Card\* card); // Nova carta para a mão

**void** ClearHand(); // Remove todas as cartas

Hand\* GetHand() // Pointer para a mão de cartas

**bool** IsBusted() // Ultrapassou a pontuação máxima

**bool** IsBlackjack() // Tem blackjack

**private**

bool ReadText(std::ifstream& **in**); // Lê de um ficheiro informações sobre o jogador

std::string \_name; // Nome

double \_balance; // Montante total

double \_bet; // Aposta

static std::string \_playersFileName; // Nome **do** ficheiro de jogadores (players.txt)

Hand\* \_hand; // Mão **do** jogador

S\_Game\* \_game; // Estado de jogo associado

## Dealer // : AllegroObject

**public**

Dealer(S\_Game\* game); // Construtor

**const** Hand\* GetHand() **const**; // Pointer para a mão de cartas

**bool** IsBusted(); // Ultrapassou a pontuação máxima

**bool** IsBlackjack(); // Tem blackjack

**void** Hit(); // Pedir

**void** Stand(); // Ficar

**void** AddNewCard(Card\* card) // Nova carta para a mão

**void** ClearHand() // Remove todas as cartas

**private**

Hand\* \_hand; // Mão do dealer

S\_Game\* \_game; // Estado de jogo associado

## State

Classe abstracta que define um estado de jogo

\* public\*

**virtual** **void** Initialize() = 0; // Inicialização de algumas variáveis

**virtual** **void** LoadContents() = 0; // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**virtual** **bool** Update(ALLEGRO\_EVENT\* ev) = 0; // Chamado cada vez que há um evento novo ; actualiza o estado

**virtual** **void** Draw() = 0; // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**virtual** **void** UnloadContents() = 0; // "Descarrega" sons, imagens e outros objectos

## S\_MainMenu : State

Classe que define o menu principal do programa

**public**

S\_MainMenu(); // Construtor

**void** Initialize(); // Inicialização de algumas variáveis

**void** LoadContents(); // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**bool** Update(ALLEGRO\_EVENT\* ev); // Chamado cada vez que há um evento novo ; actualiza o estado

**void** Draw(); // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**void** UnloadContents(); // "Descarrega" sons, imagens e botões

**private**

ALLEGRO\_BITMAP\* \_background; // Imagem de fundo

ALLEGRO\_SAMPLE\* \_bgMusic; // Música que toca **no** ínicio **do** programa

ALLEGRO\_SAMPLE\* \_nextMenuSound; // Som quando se seleciona umas das opções **do** menu

RectButton\* \_playButton; // Botão Jogar **do** menu

RectButton\* \_settingButton; // Botão Opções **do** menu

RectButton\* \_quitButton; // Botão Sair **do** menu

**int** \_selectedMenu; // Menu actualmente selecionado

## S\_Game : State

Classe que define o jogo (principal)

**public**

S\_Game(); // Construtor

**void** Initialize(); // Inicialização de algumas variáveis

**void** LoadContents(); // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**bool** Update(ALLEGRO\_EVENT\* ev); // Chamado cada vez que há um evento novo ; actualiza o estado

**void** Draw(); // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**void** UnloadContents(); // "Descarrega" sons, imagens e botões

Deck\* GetDeck() { **return** &\_deck; } // Pointer para o baralho de cartas

//- Game specific

// Events-like calls

**void** PlayerBet(Player\* player, **double** bet); // Chamado com um jogador faz bet

**void** PlayerHit(Player\* player); // Chamado com um jogador faz hit

**void** PlayerStand(Player\* player); // Chamado com um jogador faz stand

**void** PlayerDouble(Player\* player); // Chamado com um jogador faz double

**void** PlayerSurrender(Player\* player); // Chamado com um jogador desiste

**static** Player\*\* GetActivePlayers() // Retorna um array dos jogadores que estão a jogar

**static** **int** GetActivePlayerIndex() // Retorna o index do jogador activo

**void** DealerHit(Dealer\* dealer); // Chamado com o dealer faz um hit

**void** DealerStand(Dealer\* dealer); // Chamado com o dealer faz um stand

**void** HandleOutOfCards(); // Chamado quando o baralho fica sem cartas

**private**

ALLEGRO\_BITMAP\* \_background; // Imagem de fundo (mesa de jogo (verde))

Dealer\* \_dealer; // Pointer para o dealer **do** jogo

Deck \_deck; // Baralho de cartas

std::vector<Player> \_players; // Todos os jogadores

std::queue<Player\*> \_waitingPlayers; // Jogadores na fila de espera

static Player\*\* \_activePlayers; // Jogadores a jogar

**int** \_activePlayerCount; // Número de jogadores a jogar

**int** \_gameState; // Estado da ronda (apostas, vez dos jogadores, vez **do** dealer, etc.)

static **int** \_activePlayerIndex; // Index **do** jogador que está actualmente a jogar

std::vector<RectButton\*> \_buttons; // Botão **do** jogador para hit/double/stand/give up

bool HandleStatePlacingBets(); // Chamado quando **s**ão pedidas as apostas

bool HandleStateDealingCards() // Chamado com o dealer distribui as cartas

bool HandleStatePlayerTurn() // Chamado quando um jogador faz hit/double/etc.

bool HandleStateDealerTurn() // Chamado quando é a vez **do** dealer a jogar (stand/hit)

bool HandleStateResetRound(); // Reínicio da partida

bool HandleStatePostGame() { **return** true; } // Handle player surrender, etc.

void ReadPlayersFromFile(); // Lê os jogadores **do** ficheiro players.txt

void SelectPlayers(); // Seleciona os 4 primeiros jogadores da queue para jogadores activos

Player\* SelectNextPlayerFromQueue(); // Seleciona o próximo jogador que está na fila de espera (depois de um jogadores "activo" desistir)

## S\_GameOver : State

Classe que define o ecrã que é mostrado no fim do jogo. Ainda não definida completamente (baixa prioridade). É intencionado mostrar ao(s) utilizador(es) algumas estatísticas do jogo decorrido e permitir voltar ao menu principal.

**public**

S\_GameOver(); // Construtor

**void** Initialize(); // Inicialização de algumas variáveis

**void** LoadContents(); // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**bool** Update(ALLEGRO\_EVENT\* ev); // Chamado cada vez que há um evento novo ; actualiza o estado

**void** Draw(); // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**void** UnloadContents(); // "Descarrega" sons, imagens e botões

## S\_Settings : State

Classe que define o ecrã que é mostrado quando se altera as opções do jogo. Ainda não definida completamente (baixa prioridade). É intencionado permitir ao utilizador alterar a aposta mínima; alterar o volume da música/som; alterar a língua; ...

**public**

S\_GameOver(); // Construtor

**void** Initialize(); // Inicialização de algumas variáveis

**void** LoadContents(); // Carrega para memória ficheiros (sons/imagens/etc) e inicializa algumas variáveis, chamado depois de Initialize

**bool** Update(ALLEGRO\_EVENT\* ev); // Chamado cada vez que há um evento novo ; actualiza o estado

**void** Draw(); // Chamado quando é necessário desenhar o ecrã (a seguir a update)

**void** UnloadContents(); // "Descarrega" sons, imagens e botões